CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science

5003:TM:82

PIPELINED LINEAR EQUATION SOLVERS AND VLSI

Lennart Johnsson

TECHNICAL MEMORANDUM

Department of

Computer Science

CALIFORNIA INSTITUTE OF TECHNOLOGY
Computer Science
5003:TM:82

PIPELINED LINEAR EQUATION SOLVERS AND VLSI

Lennart Johnsson

Presented at
Microelectronics 1982
Adelaide, Australia
May 1982

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, ARPA Order Number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597

PIPELINED LINEAR EQUATION SOLVERS AND VLSI

S. L. Johnsson
Senior Research Associate
California Institute of Technology
Pasadena, CA 91125

1 INTRODUCTION

The communication implied by an algorithm is of particular importance in developing algorithms and architectures for VLSI. Communication is becomming the most expensive entity in terms of area and performance. Wire delays may already in todays technology limit the performance of a design if proper attention is not given to the communication aspect, [Sutherland&Mead 77], [Seitz 79]. The area of a chip design is often determined by the required wiring.

To economize the storage utilization and the requirements for arithmetic operations techniques for exploiting special properties such as symmetry and structure have been developed. Algorithms for band matrices can in a very natural way be given a for VLSI suitable communication pattern. The communication required by sparse matrix techniques does not map as easily into VLSI networks of some generality.

In this paper is discussed concurrent algorithms for the solution of linear systems of equations, the data flow implied by these algorithms, the effect of pipelining the arrays as a way to reduce the cycle time and increase the throughput, partitioning of the problems and sequencing the computations when the size of the problem does not allow for a full instantiation in space, and the complexity of the cells in computational arrays for the algorithms. The computations in different steps of a block oriented algorithm often requires a different data flow and different operations. The data flow that in an array for an algorithm fully instantiated in space is stationary becomes nonstationary. Particular attention is given to control issues. If an algorithm is fully instantiated in space the control can often be implicit in the interconnection scheme of the processing elements. When an algorithm is only partially instantiated in space and partially in time, explicit control is required. This makes the cells and the verification of correctness more complex.

The discription in this paper is made in terms of synchronouos networks. However, the algorithms can as well be implemented by asynchronouos networks.

2 METHODS FOR SYSTEMS OF LINEAR EQUATIONS

Iterative as well as direct methods are used to solve linear systems of equations. Which method that is used depends on several factors such as structure and size of the matrices, the data and required accuracy, see e.g. [Dahlquist 74], [Wilkinson 65]. Common iterative methods are Gauss-Jacobi, Gauss-Seidel and in particular relaxation methods. A direct approach to concurrency for iterative methods is to employ a computational network that directly mimics the graph that corresponds to the matrix. If the matrix is the result of difference approximation of partial differential equations a full instantiation in space would have one processing element per mesh point and direct connnection to a few neighboring elements. The interconnections are a direct reflection of the approximation formula being used. For a full matrix the graph is complete and the number of interconnections per processing element unrealisticly large for large matrices.

Direct methods leads to a solution in a fixed number of steps. Often, no information about the solution is available before the final step of the method. Direct methods for the solution of linear equations are Gaussian elimination and the related methods due to Doolittle, Crout and Cholesky, all leading to a LU-decomposition of the system matrix as an intermediate step in the solution process. Other direct methods are Householder transformations and Given's rotations, both leading to a QR-decomposition as an intermediate step. The matrices U and R are both upper triangular. Yet another direct method is the conjugate gradient method devised by Hestenes and Stiefel.

The triangular factors that are intermediate results in the solution process can be stored and used to solve any number of equations with the same system matrix. For sparse matrices a particular advantage of factorizing the system matrix is that the factors are in general still sparse, although typically less sparse than the original matrix, whereas the inverse in general is a full matrix.

Concurrent algorithms for Gaussian elimination has been presented by H. T. Kung [Kung&Leiserson 80] and S.Y. Kung [Kung 80]. H. T. Kung uses hexagonally interconnected processing elements. S.Y. Kung presents his algorithms for elements organized in an orthogonal mesh interconnected to have the same communication capabilities as the hexagonal arrays. Hexagonal interconnect is ideal for Gaussian elimination with pivoting along the diagonal. In both the above mentioned references a full instantiation in space is assumed. The discussion here of concurrent Gaussian elimination focuses on algorithms only partially instantiated in space and pipelining of the corresponding arrays.

We also present algorithms and arrays for Householder transformations and Given's rotations, both numerically stable methods, [Wilkinson 65]. A different concurrent algorithm for Given's method has been presented by Gentleman and Kung, [Gentlemen&Kung 81]. An algorithm for Given's method similar to the one discussed in this paper is proposed by Heller and Ipsen, [Heller&Ipsen 82].

The algorithms are described either in standard mathematical notation or by means of graphs, which gives an immediate understanding of the concurrency in the algorithm and also suggests a way of implementation. When a full instantiation in space is unrealistic a graph is still a good illustration of the static features of the network, but do not reflect the dynamic behavior and the control. A good way to formally derive and verify concurrent algorithms is highly desirable.

3 GAUSSIAN ELIMINATION

Given a system of equations

$$Ax = y$$

where A is a nonsingular, N by N matrix and x and y vectors, Gaussian elimination transforms the system of equations into a triangular linear system

$$Ux = L^{-1}y$$

that is solved by back substitution.

Algorithms exploiting concurrency at the vector level will now be discussed. Additional concurrency is possible for band matrices and sparse matrices but is not considered here, see e.g. [Johnsson 80].

Assuming that variables 1 through k-1 have been eliminated the next step is to eliminate variable k. Denoting the matrix that is the result of the first k-1 variable eliminations A^{k-1} the computations that are necessary to obtain A^k are

$$A^k(i,j) = A^{k-1}(i,j) - A^{k-1}(i,k) * A^{k-1}(k,j) / A^{k-1}(k,k) \quad k+1 \leq i,j \leq N$$

assuming $A^{k-1}(k,k)$ is nonzero.

From this expression it is apparent that the computations for the different rows and columns are independent of each other. Hence, all these computations can be performed concurrently. For a full matrix (N-1)x(N-1) inner product cells are required for full concurrency in the first elimination step, (N-2)x(N-2) such cells in the second elimination step, etc. In the particular case where A is a band matrix operations outside the band are trivial and requires no cells. With r co-diagonals below the main diagonal and s co-diagonals above the main diagonal a total of rx(s+1) cells are required for the first N-s steps of the elimination. The computationally active part can be represented by a "window" that slides down the diagonal, Figure 1.

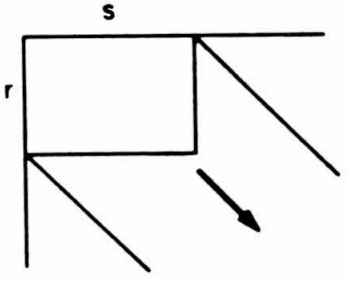

Figure 1: Computational window for a band matrix

In the most direct implementation of the computations defined above the uppermost row of the computational window is broadcasted to the other rows and the factors multiplying that row are broadcasted to the columns. Such an implementation requires N-1 cycles for the elimination. The length of a cycle may have to be substantial. The broadcasting can be avoided by introducing intermediate storage, denoted Z, between the different rows and columns, so that the uppermost row progresses downwards one row per cycle and the row multilication factors move to the right one column per cycle. The computations associated with the elimination of the next variable can start when the required diagonal element, say $A^k(k+1,k+1)$, has been computed. Hence, several steps can be pipelined. To keep the same processing elements busy the computational window is "fixed" in space and the data made to flow through the cells as is shown in Figure 2.

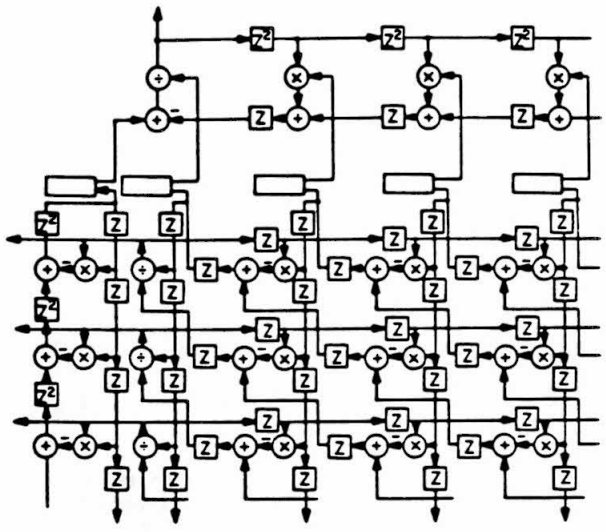

Figure 2: A pipelined array for Gaussian elimination

There are three directions of data flow in the array in Figure 2, left to right, top to bottom, and diagonal. The flow corresponds to hexagonal interconnect. Due to the interaction between the different data streams some data items flows in loops making the flow turbulent. The innermost loops that a data item traverses have a few register cells in them in addition to the arithmetic units. A number of clock cycles has to elapse before the data in updated form is back in the same location. In the array in Figure 2 three cycles are required for the array to progress from any state for one row to the same state for the next row. Hence, in the pipelined array 3(N-1) cycles are required to perform the triangulation.

There are three directions of data flow in the array in Figure 2, left to right, top to bottom, and diagonal. The flow corresponds to hexagonal interconnect. Due to the interaction between the different data streams some data items flows in loops making the flow turbulent. The innermost loops that a data item traverses have a few register cells in them in addition to the arithmetic units. A number of clock cycles has to elapse before the data in updated form is back in the same location. In the array in Figure 2 three cycles are required for the array to progress from any state for one row to the same state for the next row. Hence, in the pipelined array 3(N-1) cycles are required to perform the triangulation.

It should be noticed that there is no central control of the array in Figure 2. The control is distributed and pipelined in the same manner as the computations. Neighboring cells are either one step ahead or one step behind each other.

The operations on the right hand side of the linear system of equations are the same as on the last column of A and can be performed concurrently with operations on A. Several right hand sides can be treated concurrently provided enough cells are available. The computations on the right hand side are in Figure 2 placed to the left of the computations associated with A. The elimination and forward substitution has to be completed before the back substitution starts. The upper triangular matrix is computed with row one first and row N last, while in the back substitution row N is needed first and row one last. The diagonal of the upper triangular matrix U is stored in the leftmost stack above the elimination part of the array, while the nonzero co-diagonals above the main diagonal are stored in subsequent stacks to the right, one co-diagonal per stack.

The simple form of Gaussian elimination described above works in many cases, in particular where A is symmetric positive definite or diagonnaly dominant. Partial pivoting, total pivoting, neighbor pivoting or threshold pivoting may be required to assure sufficient accuracy of the result. Partial pivoting is in most cases but not always sufficient to achieve acceptable accuracy. Partial pivoting causes a change in the data flow in the array corresponding to changing or creating pointers in sequential programs. Pivoting on an off diagonal element will in general increase the bandwidth of the matrix. The data flow when pivoting is performed on or off the diagonal is shown in Figure 3.

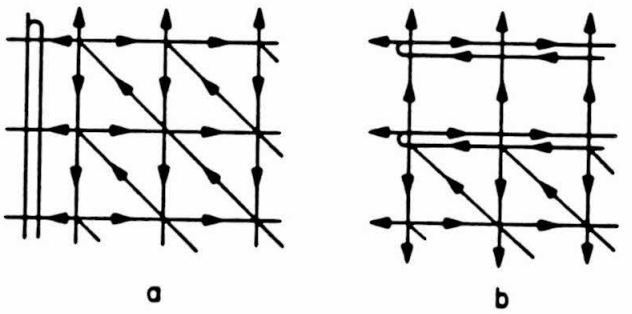

Figure 3: Stream lines, Gaussian elimination

Changing the direction of the data flow causes a loss of performance, as is typical for any pipeline. This characteristic is easy to see from the wave fronts of the computations. A wave front can be considered as consisting of points on different stream lines that are of the same phase. The wave fronts show the necessary synchronization between different data streams. Figure 4 shows the wave fronts when pivoting is performed on and off the diagonal, respectively.

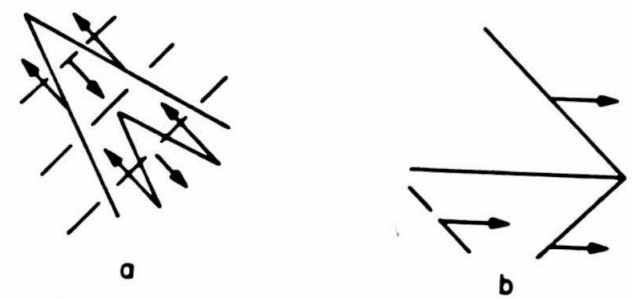

Figure 4: Wave fronts, Gaussian elimination

From Figure 4 it should be clear that on the average (r/2+3) cycles could be lost for each column elimination, turning the algorithm into one of O(rxN) time complexity. The additional cycles are required to allow for the elements in the pivot column to be computed and the computations restarted in the proper location. At most two cells in a row or a column will be active concurrently. The computations essentially becomes one dimensional.

If the algorithm can not be fully instantiated in space, block algorithms can be used. Each block shall be of a size sufficiently small to be treated within the array concurrently. Pivoting shall for a block oriented array be performed outside the array, since otherwise a substantial performance penalty may have to be payed. In a block oriented algorithm the diagonal blocks are factorized as previously discussed while the off diagonal blocks above the diagonal undergo the same operations as the right hand side, while the blocks below the diagonal have to be treated slightly different. An example of a block algorithm is given below.

Step 1 $\quad L_{ii}^{i} U_{ii}^{i} = A_{ii}^{i-1} \qquad y_i^i = (L_{ii}^{i})^{-1} y_i^{i-1}$

Step 2 $\quad L_{ii}^{i-1} A_{ij}^{i-1} = U_{ij}^{i}$

Step 3 $\quad L_{ji}^{i} U_{ii}^{i} = -A_{ji}^{i-1} \qquad y_j^i = y_j^{i-1} - L_{ji}^{i} y_i^i$

Step 4 $\quad A_{kj}^{i} = A_{kj}^{i-1} - L_{ki}^{i} U_{ij}^{i}$

The array presented earlier has to be modified in order to accomodate the set of computations defined by the block algorithm. New paths for data through the array are necessary. Four different flow patterns are sufficient for partial pivoting and the block algorithm. The different steps of the block algorithm can be pipelined. Since the data flow within the array is different during the different steps of the algorithm data can be accepted every second cycle in some steps and every cycle in others instead of every third cycle as in the factorization of the diagonal blocks. For a detailed analysis see Johnsson, [Johnsson 81]. A cell in the elimination part of the array in Figure 2 that is general enough also to serve block algorithms is shown in Figure 5.

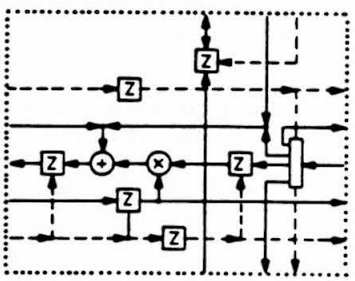

Figure 5: Elimination cell

4 HOUSEHOLDER TRANSFORMATIONS

Housholder's method has several similarities with Gaussian elimination from a computational point of view. The difference lies in the way the pivot row is selected or rather computed. Instead of selecting a pivot element and using the row corresponding to it in the elimination process a "pivot row" is computed as a normalized linear combination of the rows in the untriangulated part of the matrix. The coefficients in the linear combination are the elements of the first column of the part of the untriangulated matrix divided by a

normalization factor. This particular selection of coefficients means that only rows with nonzero entries in the first column are used in the computation of the pivot row. The normalization is made so that the transformation made in each step is unitary. The equations defining the Householder transformation can be written as follows:

$$P_j = I - 2w_j w_j^* \qquad w_j^* w_j = 1$$

$$u_k = 0 \qquad k = 1,2,\ldots,j-1$$

$$u_j = A^j(j,j) \pm s$$

$$u_k = A^j(k,j) \qquad k = j+1, \ldots, N$$

$$s = |u|$$

$$w_j = u/\sqrt{2s^2 \pm 2A^j(j,j)s}$$

$$A^j = P_{j-1} P_{j-2} \cdots P_1 A$$

The normalization implies that the elimination step for column k-1 has to be completed before the elimination step for column k can be initiated. In order to compute the normalization factor required in the elimination of column k all the elements in that column need to be known. However, it is possible to initiate the computation of the normalization factor as soon as row k has been computed in the elimination of column k-1. Hence, the elimination of two successive variables can be performed concurrently and pipelined. Operations on different columns can also be performed concurrently.

A concurrent algorithm for Householder transformations is given in [Johnsson 82]. A graphic illustration showing the flow of data and the operations thereon is given in Figure 6. The boundary cell to the left computes the normalization factor. It is followed by r+s-1 identical internal cells for elimination and "pivot row" computation. The right boundary cell can be simplified since the "pivot row" computations only will include one element. The control of the array is periodic with a periodicity corresponding to the length of a column plus a few cycles for normalization. The sequential aspects are shown in Figure 7.

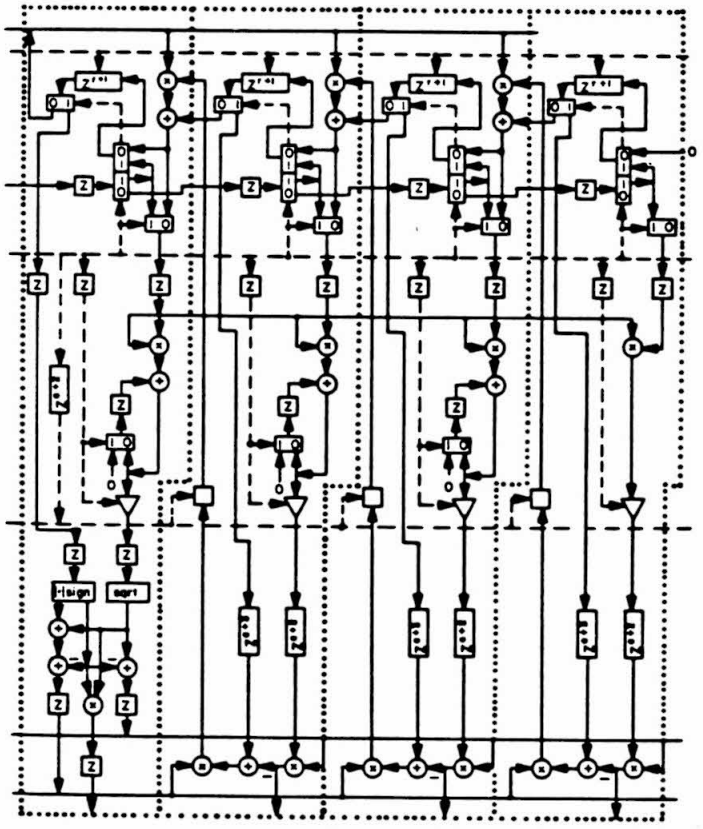

Figure 6: An array for Householder transformations

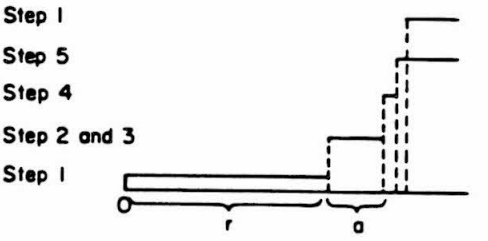

Figure 7: Sequencing of computations in Figure 6

The time for triangulation is (r+a+3)N cycles, where the value of a equals the number of cycles required for the computation of the square root, and the number 3 is due to pipelining. For a discussion of performance related issues see [Johnsson 82].

5 GIVEN'S METHOD

In Given's method of plane rotations two rows are the subject of rotation in each step. A rotation operation is performed so that the element in the first column of one of the rows become zero.

$$\begin{bmatrix} u(1,1) & u(1,2)\ldots u(1,N) \\ 0 & u(2,2)\ldots u(2,N) \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} v(1,1)v(1,2)\ldots v(1,N) \\ v(2,1)v(2,2)\ldots v(2,N) \end{bmatrix}$$

$$d = \sqrt{v(1,1)^2 + v(2,1)^2}$$

$$c = v(1,1)/d \qquad s = v(2,1)/d$$

There are several ways of exploiting the concurrency in Given's method. Obviously operations on different columns are independent and can be performed concurrently. To eliminate the nonzero entries below the diagonal in a column several rotations are necessary. A concurrent algorithm with many similarities with the algorithm for Gaussian elimination presented earlier is to propagate the top row down through the array. However, unlike in Gaussian elimination the row that is propagated through the array also has to be changed. Hence, in each cell four multiplications and two additions should be performed.

The upper triangular matrix U and the result of the forward substitution are obtained at the bottom of the array instead of the top. The rotations associated with successive columns can be pipelined in the same way as for Gaussian elimination. Hence the main data flow is the same but more operations have to be performed. The time to complete the triangulation depends on how pipelining is implemented but is of the form aN+r where a is less than 10. The number of processors in the triangulation part of the array is ideally equal to rx(r+s/2+1).

The boundary processors of the triangulation part of the array do not perform a simple divide as in Gaussian elimination, but should compute the rotation factors which includes square roots. There exist variations of Givens method that doesn't

require square roots, [Gentlemen 73], [Hammarling 74]. Instead a few, e.g. 2-5 multiplications and 1-2 divisions are performed.

An algorithm similar to the one above has been described by Heller and Ipsen, [Heller&Ipsen 82]. Another algorithm in which the rows become stationary in the array when the transformations they undergo are completed is described by Gentleman and Kung, [Gentlemen&Kung 81]. At the end the array contains the upper triangular matrix R. For a band matrix the array should in this case contain $(r+s+1)N-(r+s+1)(r+s+1)/2$ processors.

6 CONCLUSION

There are several similarities between Gaussian elimination, Householder transformations and Given's rotations. The data flow in Gaussian elimination without partial pivoting and Given's rotations is essentially the same but the latter requires four multiplications and two additions compared to one multiplication and one addition for the former method. Gaussian elimination without partial pivoting uses fewer cells than Given's method.

In Householder's method the number of cells used equals $(r+s+1)$ which is a factor of r less than for Given's method. Instead the solution time becomes proportional to $(r+a+3)N$. For large values of r the difference is significant. The cells used in Householder's method are essentially of the same complexity as those required in Given's method. Gaussian elimination with partial pivoting should on the average require $(3r/2)N$ cycles, at worst $2rN$ and at best rN cycles. Hence, depending on r Householder's method may be faster than Gaussian elimination with partial pivoting. However, a linear array for Gaussian elimination with partial pivoting can be built with smaller cells.

Pipelining of the arrays discussed here is used to decrease the cycle time with the intent to increase the total throughput. However, due to the turbulent nature of the data flow, it is necessary to spread data out in time a number of cycles that corresponds to the number of delays in the innermost loops.

In all the arrays the control is distributed in the same manner the computations are. There is no global controller. The control phase of neighboring cells differs by one in the pipelined arrays. The control of the data flow in space and time requires only a few bits for the methods discussed here.

The similarity in data flow and control as well as the arithmetic requirements for the different methods suggests that a general cell of limited complexity should be able to efficiently serve all three methods. Such a cell should also be useful for computations such as matrix multiplications, eigenvalue computations and the computation of the discrete Fourier transform.

7 ACKNOWLEDGEMENT

The author gratefully acknowledges the support provided by the Defense Advanced Research Project Agency under contract N00014-79-C-0597 with the California Institute of Technology. Views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion of DARPA, the U.S. Government, nor any person or agency connected with them.

8 REFERENCES

[Dahlquist 74] Dahlquist G., Bjorck A., Anderson N., Numerical Methods, Prentice-Hall, 1974.

[Gentlemen&Kung 81] Gentlemen, S. Morven, Kung H. T., Matrix Triangulation by Systolic Arrays, Carnegie-Mellon University, Technical Report, 1981.

[Gentlemen 73] Gentlemen, S. Morven, "Least Squares Computation by Givens Transformations without Square Roots," J. Inst. Maths. Applics. 12, 1973, 329-336.

[Hammarling 74] Hammarling, S., "A Note on Modifications to the Givens Plane Rotation," J. Inst. Maths. Applics. 13, 1974, 215-218.

[Heller&Ipsen 82] Heller, Don E., Ipsen, Ilse I. P., "Systolic Networks for Orthogonal Equivalence Transformations and Their Applications," in Advanced Research in VLSI, MIT, 1982.

[Johnsson 80] Johnsson, Lennart, Gaussian Elimination on Sparese Matrices and Concurrency, Caltech Computer Science Department, Technical Report 4087, December 1980.

[Johnsson 81] Johnsson, L., Computational Arrays for Band Matrix Equations, Caltech Computer Science Department, Technical Report 4287, May 1981.

[Johnsson 82] Johnsson L., "A Computational Array for the QR-method," in Advanced Research in VLSI, MIT, January 1982.

[Kung&Leiserson 80] Kung, H.T. and Leiserson, Charles E., "Algorithms for VLSI Processor Arrays," in Introduction to VLSI Systems, Addison-Wesley, 1980. Mead, Carver A. and Conway, Lynn A.

[Kung 80] Kung, S.Y., "VLSI Matrix Computation Array Processor," in MIT Conference on Advanced Research in Integrated Circuits, MIT, February 1980.

[Seitz 79] Seitz Charles L., "Self-Timed VLSI Systems," in The Caltech Conference on VLSI, Caltech, 1979.

[Sutherland&Mead 77] Sutherland Ivan, Mead Carver A., "Microelectronics and Computer Science," Scientific American 237, (3), September 1977, 210-229.

[Wilkinson 65] Wilkinson, J. H., The Algebraic Eigenvalue Problem, Oxford, 1965.

ABSTRACT

Many of the commonly used methods for solution of linear systems of equations on sequential machines can be given a concurrent formulation. The concurrent algorithms take advantage of independence of operations in order to reduce the time complexity of the methods. During the course of computations specified by the algorithm data has to be routed to the various places of computation. Pipelining can be used to avoid broadcasting in VLSI arrays for computation. Pipelining will in general allow for a reduced cycle time but may force data to be spread out in time, as is the case for Gaussian elimination. What the required spacing is depends on the pipelining and the data flow.

In the paper concurrent algorithms and their pipelining for Gaussian elimination, Householder transformations and Given's rotations are discussed. Gaussian elimination and Given's rotations can use two dimensional arrays while Householder transformation uses a one dimensional array. If partial pivoting is necessary in Gaussian elimination, then one dimension of the array is essentially lost and a linear array is almost as efficient as a two-dimensional array. Householder transformations that are numerically stable may perform the triangulation in shorter time, if partial pivoting is necessary in Gaussian elimination. The amount of arithmetic that a node in the arrays perform is somewhat different for the different methods. The difference is largest for the boundary cells. However, it should be feasible to design a common node of very low complexity that very efficiently supports a range of methods for the solution of linear systems of equations.

Keywords: Concurrent algorithms
Numerical analysis
Computational Arrays
VLSI